4. IMPLEMENTASI SISTEM

Bab ini menguraikan implementasi desain sistem yang telah dirumuskan pada bab sebelumnya. Implementasi ini dilakukan dengan menggunakan bahasa pemrograman Javascript dan *framework* Node.js untuk pengembangan aplikasi, serta JSON file sebagai basis data. Adapun tools yang digunakan dalam proses implementasi ini adalah Visual Studio Code dan Terminal. Tabel 4.1 merupakan list implementasi diagram aktivitas ke dalam segmen program.

Tabel 4.1 Implementasi
Diagram aktivitas kedalam segmen program

Activity Diagram	Segmen Program	Keterangan
Gambar 3.1	Segmen 4.7	Fungsi Mengirim Pesan
	Segmen 4.9	Fungsi Handler Transaksi
	Segmen 4.10	Fungsi Verifikasi Transaksi
	Segmen 4.11	Fungsi Transaksi Digiflazz
	Segmen 4.16	Fungsi Verifikasi ID
	Segmen 4.17	Fungsi Error Handler
Gambar 3.2	Segmen 4.12	Fungsi Update Produk
	Segmen 4.13	Fungsi Isi Saldo Digiflazz
	Segmen 4.14	Fungsi Cek Saldo Digiflazz
	Segmen 4.15	Fungsi Cek Saldo LinkQu

4.1. Spesifikasi Instance

Deployment bot Whatsapp menggunakan layanan VPS dari *DigitalOcean* dengan spesifikasi sebagai berikut :

- Tipe : Basic Droplet – Regular CPU
- Operating System : Ubuntu 22.04 LTS
- Processor / vCPU : 2 cores
- RAM : 2 GiB
- Storage : 60 GiB SSD
- Bandwidth : 3,000 GiB Transfer

4.2. Konfigurasi *environment*

Untuk alasan keamanan dan pengelolaan kredensial aplikasi yang lebih baik, implementasi sistem ini memanfaatkan file ".env" (dot-env) untuk menyimpan variable-variable sensitif. Variable sensitif ini meliputi kredensial API, token akses, dan informasi rahasia lainnya yang tidak boleh ter ekspos dalam kode sumber maupun didistribusikan secara publik.

Segmen Program 4.1 Konfigurasi file ".env" (dot-env)

```
NAMA_BOT=crowdbytehbotol
DIGIFLAZZ_USERNAME=
DIGIFLAZZ_PRODUCTION_KEY=
LINKQU_USERNAME=
LINKQU_PIN=
LINKQU_SECRET_KEY=
LINKQU_CLIENT_ID=
LINKQU_CLIENT_SECRET=
```

4.3. Konfigurasi Sistem dan Pengaturan Global

File config.js berperan penting dalam mengkonfigurasi berbagai aspek sistem, termasuk variabel global, pengaturan koneksi API, parameter modal, detail API game, dan informasi rekening pembayaran. Analisis terhadap file ini akan memberikan pemahaman yang lebih mendalam mengenai konfigurasi sistem dan interaksinya dengan komponen lain.

Segmen Program 4.2 Konfigurasi file "config.js"

```
const fs = require('fs');
global.pemilik = ['6282227114586']; // NOMER PEMILIK
global.namaSesi = 'CROWDBYTEHBOTOL'; // NAMA SESI
global.message = {
  sukses: 'Berhasil',
  proses: 'Transaksi anda sedang diproses, silahkan klik refresh untuk update status transaksi anda',
  gagal: 'Gagal',
  maintenance: 'Sistem sedang maintenance ',
  idSalah: 'Data tidak ditemukan ',
global.modals = {
  persen: 0.02, // MENGATUR MARKUP ATAU KEUNTUNGAN
  markupMin: 0.02, // MENGATUR MARKUP ATAU KEUNTUNGAN MINIMAL
```

```
  markupMax: 0.02, // MENGATUR MARKUP ATAU KEUNTUNGAN MAKSIMAL
  biayaQris: 0.009, // MENGATUR BIAYA QRIS
};
global.apigames = {
  member: 2670,
  vip: 2620,
  merchant: '',
  secret: '',
};
```

4.4. Implementasi

4.3.1 Fungsi Bot Virtual

Segmen Program 4.3 *Function virtual bot*

```
async function startVirtualbot() {
  const { state, saveCreds } = await useMultiFileAuthState(`session`);
  const vbot = virtualbotConnect({
    logger: pino({ level: 'silent' }),
    printQRInTerminal: true,
    browser: [`${namaSesi}`, 'Safari', '1.0.0'],
    patchMessageBeforeSending: (message) => {
      const requiresPatch = !!( message.buttonsMessage || message.templateMessage ||
        message.listMessage
      );
      if (requiresPatch) {
        message = {
          viewOnceMessage: {
            message: {
              messageContextInfo: {
                deviceListMetadataVersion: 2,
                deviceListMetadata: {},
              },
              ...message,
            },
          },
        };
      }
      return message;
    },
    auth: state,
```

```
});
title();

store.bind(vbot.ev);
```

Fungsi *startVirtualBot* merupakan fungsi *asynchronous* yang digunakan untuk memulai bot virtual. Fungsi ini diawali dengan mendapatkan status authentikasi yang tersimpan pada folder *session* menggunakan fungsi *useMultiFileAuthState.* Kemudia, fungsi ini membuat koneksi ke bot virtual dengan menggunakan fungsi *virtualBotConnect*. Saat terhubung, fungsi ini akan mengatur beberapa hal seperti menonaktifkan *logger*, menamplikan kode QR di terminal, mengatur browser yang akan digunakan, dalam hal ini *Safari* dengan versi 1.0.0, melakukan *patch* terhadap pesan yang akan dikirimkan sebelum proses pengiriman. *Patch* ini bertujuan untuk mengubah format pesan menjadi sekali lihat (*view once message*).

Segmen Program 4.4 Memory Store

```
const store = makeInMemoryStore({
   logger: pino().child({ level: 'silent', stream: 'store' }),
});
```

Variable *store* merupakan objek yang merepresentasikan penyimpanan data internal untuk bot virtual. Dibuat dengan menggunakan fungsi *makeInMemoryStore.* Fungsi tersebut memungkinkan penyimpanan data di dalam memori, tanpa perlu menyimpannya ke file secara permanen. Tujuan utama *store* adalah untuk menyimpan informasi yang dibutuhkan bot virtual selama beroperasi, seperti informasi autentikasi, status pesan, konfigurasi bot, dan lain sebagainya. Penggunaan penyimpanan internal ini memungkinkan bot virtual untuk mengakses data dengan cepat dan efisien, tanpa perlu mengandalkan penyimpanan eksternal yang kemungkinan lebih lambat.

4.3.2 Fungsi Informasi Kontak

Segmen Program 4.5 Menyimpan informasi kontak pada variable *store*

```
vbot.decodeJid = (jid) => {
   if (!jid) return jid;
   if (/:\d+@/gi.test(jid)) {
      let decode = jidDecode(jid) || {};
      return (
         (decode.user &&
            decode.server &&
```

```
            decode.user + '@' + decode.server) ||
          jid
        );
    } else return jid;
};

vbot.ev.on('contacts.update', (update) => {
    for (let contact of update) {
        let id = vbot.decodeJid(contact.id);
        if (store && store.contacts)
            store.contacts[id] = { id, name: contact.notify };
    }
});

vbot.getName = (jid, withoutContact = false) => {
    id = vbot.decodeJid(jid);
    withoutContact = vbot.withoutContact || withoutContact;
    let v;
    if (id.endsWith('@g.us'))
        return new Promise(async (resolve) => {
            v = store.contacts[id] || {};
            if (!(v.name || v.subject)) v = vbot.groupMetadata(id) || {};
            resolve(
                v.name ||
                    v.subject ||
                    PhoneNumber(
                        '+' + id.replace('@s.whatsapp.net', '')
                    ).getNumber('international')
            );
        });
    else
        v =
            id === '0@s.whatsapp.net'
                ? {
                      id,
                      name: 'WhatsApp',
                  }
                : id === vbot.decodeJid(vbot.user.id)
                ? vbot.user
                : store.contacts[id] || {};
    return (
        (withoutContact ? '' : v.name) ||
        v.subject ||
        v.verifiedName ||
```

```
        PhoneNumber('+' + jid.replace('@s.whatsapp.net', '')).getNumber(
          'international'
        )
    );
};
vbot.sendContact = async (jid, kon, quoted = '', opts = {}) => {
    let list = [];
    for (let i of kon) {
        list.push({
            displayName: await vbot.getName(i + '@s.whatsapp.net'),
            vcard: `BEGIN:VCARD\nVERSION:3.0\nN:${await vbot.getName(
                i + '@s.whatsapp.net'
            )}\nFN:${await vbot.getName(
                i + '@s.whatsapp.net'
            )}\nitem1.TEL;waid=${i}:${i}\nitem1.X-ABLabel:Ponsel\nEND:VCARD`,
        });
    }
    vbot.sendMessage(
        jid,
        {
            contacts: {
                displayName: `${list.length} Kontak`,
                contacts: list,
            },
            ...opts,
        },
        { quoted }
    );
};

vbot.public = true;

vbot.serializeM = (m) => smsg(vbot, m, store);
```

Kode ini berfokus pada manajemen kontak dan informasi *user*. Kode ini memanfaatkan variable *store* untuk menyimpan informasi kontak. Ketika menerima update kontak melalui *event contacts.update*, detail kontak seperti ID dan nama panggilan (*notify*) disimpan kedalam objek *store.contacts*. Selain itu fungsi *vbot.decodeJid* digunakan untuk memvalidasi dan men-*decode* format ID *user* sebelum disimpan. Terakhir, fungsi *vbot.serializeM* digunakan untuk memproses pesan yang diterima dangan memanfaatkan informasi kontak yang tersimpan di dalam *store*.

4.3.3 Fungsi status koneksi

Untuk menjaga kelancaran operasi bot virtual, diperlukan fungsi untuk memantau status koneksi dan kredensial, agar jika terjadi *error* seperti koneksi internet terputus, bot virtual dapat melakukan koneksi ulang atau melakukan authentikasi ulang dalam beberapa kasus seperti akun telah *logout* atau file *session* tidak ditemukan.

Segmen Program 4.6 Fungsi status koneksi

```
vbot.ev.on('connection.update', async (update) => {
    const { connection, lastDisconnect } = update;
    if (connection === 'close') {
        // reconnect if not logged out
        if (
            new Boom(lastDisconnect?.error)?.output.statusCode !==
            DisconnectReason.loggedOut
        ) {
            startVirtualbot();
        } else {
            console.log(
                chalk.redBright(
                    '\nKoneksi Ke Server Terputus... Device Logged Out.'
                )
            );
        }
    }
    console.log(
        chalk.green(
            `Koneksi Ke Server Terhubung, Bot ${namaSesi} Siap Digunakan.`
        )
    );
});

vbot.ev.on('creds.update', saveCreds);
```

Event connection.update merupakan *event listener* yang digunakan untuk memeriksa status koneksi. Jika koneksi ditutup, langkah selanjutnya adalah menentukan penyebabnya, jika koneksi tertutup bukan karena *session* telah di *logout*, maka bot virtual akan mencoba untuk terhubung kembali ke server dengan memulai ulang bot virtual dengan menjalankan fungsi *startVirtualBot.* Jika terjadi logout maka akan ditampilkan pesan bahwa perangkat telah *logout* untuk menginformasikan *user.* Selain itu pada segmen program ini juga dilakukan *event creds.update* dan memanggil fungsi *saveCreds* untuk menyimpan kredensial bot virtual. Hal ini digunakan untuk memastikan bot virtual dapat terus beroperasi dengan menggunakan kredensial yang terbaru dan valid.

4.3.4 Fungsi mengirim pesan

Segmen Program 4.7 Fungsi mengirim pesan

```
vbot.sendText = (jid, text, quoted = '', options) =>
  vbot.sendMessage(
    jid,
    { text: text, ...options },
    { quoted, ...options }
  );
vbot.sendImage = async (jid, path, caption = '', quoted = '', options) => {
  let buffer = Buffer.isBuffer(path)
    ? path
    : /^data:.*?\/.*?;base64,/i.test(path)
    ? Buffer.from(path.split`,`[1], 'base64')
    : /^https?:\/\//.test(path)
    ? await await getBuffer(path)
    : fs.existsSync(path)
    ? fs.readFileSync(path)
    : Buffer.alloc(0);
  return await vbot.sendMessage(
    jid,
    { image: buffer, caption: caption, ...options },
    { quoted }
  );
};
```

Kode ini digunakan untuk fungsi mengirim pesan teks dan gambar. Fungsi *vbot.sendText* memungkinkan pengiriman pesan teks dasar dengan menyertakan ID penerima, isi pesan dan opsi tambahan, dan fungsi *vbot.sendImage* digunakan untuk pengiriman gambar dengan menyertakan ID penerima, lokasi gambar (*path*), *caption*, dan opsi tambahan. Path dalam fungsi ini bisa berupa *buffer* yang sudah berisi data gambar, format *base64* yang dikonversi menjadi *buffer*, URL yang gambarnya akan diunduh, atau file yang dibaca menggunakan modul *fs.* Hasil dari kedua fungsi ini akan diteruskan (*return*) ke fungsi *vbot.sendMessage* untuk mengirimkan pesan kepada ID penerima.

4.3.5 Fungsi Handler User Input

Segmen Program 4.8 Fungsi Handler User Input

```
const findState = (from) => {
    return userState.find((obj) => obj.from === from)?.state;
  };

  const updateState = (from, state) => {
    const existingFrom = userState.find((obj) => obj.from === from);
    if (existingFrom) {
      existingFrom.state = state;
    } else {
      userState.push({ from, state: state });
    }
  };

  switch (command) {
    default:
      if (findState(from) === undefined || findState(from) === null) action = 'menu';
      else if (findState(from) === 2) {
        action = list[command - 1];
      } else if (findState(from) === 3) action = command;
      else updateState(from, 1);
      break;
    case '1':
      if (findState(from) === 1) action = 'topupgames';
      break;
    case '2':
      if (findState(from) === 1) action = 'owner';
      break;
    case '99':
      if (findState(from) > 1) userState[from].state -= 1;
  }
```

Kode ini digunakan untuk *handling* input yang dikirimkan oleh user saat berinteraksi dengan menu bot dan menentukan posisi level (*state*) setiap user pada menu saat itu. Posisi setiap user akan disimpan dalam *userState*, dan untuk memindahkan posisi user digunakan fungsi *updateState*.

4.3.6 Fungsi Handler Transaksi

Segmen Program 4.9 Fungsi Handler Transaksi

```
{
        if (konek.DF !== 'YA') {
          m.reply(mess.maintenance);
        } else {
          const sku = text.split(' ')[0];
          const tujuan = text.split(' ')[1];
          const refId = `TRX${nanoid()}`;
          const listHarga = produkDF;
          const filteredList = listHarga
            .filter((item) => item.buyer_sku_code === sku)
            .sort((a, b) => a.price - b.price);
          if (filteredList.length === 0) {
            m.reply(`Produk dengan SKU *${sku}* tidak ditemukan. Silahkan cek kembali SKU produk
yang Anda masukkan`);
          } else {
            const product = filteredList[0];
            const harga =
              product.price < 0
                ? product.price + xjual.markupMin
                : product.price + Math.ceil(product.price * xjual.persen);
            const margindf = harga - product.price;
            const biaya = Math.ceil(parseInt(harga) * modals.biayaQris);
            const jumlahIsi = parseInt(harga) - biaya;
            const nama = chance.name();
            const email = chance.email({ domain: 'gmail.com' });
            const depoRefId = `DEPO${nanoid()}`;
            const cusId = `${nanoid()}`;
            if (product.seller_product_status === false) {
              m.reply('produk sedang off silahkan hubungi admin');
              vbot.sendMessage(`${pemilik}@s.whatsapp.net`, { text: laporaner }, { quoted: m });
            } else {
               const hsl = await koneksiLQ.qrisLQ(
                 harga,
                 depoRefId,
                 cusId,
                 nama,
                 kadaluarsaLQ,
                 ubahNoHp(sender),
                 email
               );
```

```
            const batasBayar = moment(hsl.expired, 'YYYYMMDDHHmmss').format('DD/MM/YYYY, HH:mm:ss z');
            const desk = `*「 KONFIRMASI PEMBAYARAN 」*

Silahkan cek data berikut ini :

${tanda} Produk : ${product.product_name}
${tanda} ID transaksi : ${hsl.partner_reff}
${tanda} Batas waktu pembayaran : *${batasBayar}*
${tanda} Jumlah bayar : *${rp(harga)}*
${tanda} Potongan biaya layanan : ${rp(biaya)}
××××××××××××××××××
Jika sudah melakukan pembayaran, silahkan copy dan paste format yang telah disediakan untuk update status transaksi anda

${botName}`;
            vbot.sendMessage(
              from,
              {
                image: { url: hsl.imageqris },
                caption: desk,
              },
              { quoted: m }
            );
```

Kode ini digunakan untuk membuat transaksi setelah *user* mengirimkan pesan berupa format yang telah ditentukan. Pada fungsi ini akan dilakukan verifikasi apakah format pesan telah sesuai dan *stock* produk tersedia. Setelah terverifikasi, pada fungsi ini dilakukan kalkulasi harga yang harus dibayarkan oleh pembeli. Fungsi *koneksiLQ.qrisLQ* digunakan untuk melakukan *generate* kode QR berdasarkan informasi yang diberikan. Semua informasi yang telah dirangkum di kirimkan kepada pembeli.

Segmen Program 4.10 Fungsi verifikasi transaksi

```
const qrbrolq = await ceklqbro(hsl.partner_reff);
 if (qrbrolq.status_trx === 'succes') {
  const hasil = await cekTrxDF(sku, tujuan, depoRefId);
  console.log(hasil);
  const db = readDbRiwayatTopup();

  for (const i of db) {
   if (depoRefId === i.id) {
   if (i.status === 'Sukses') {
   const sukses = `* 「 TOPUP BERHASIL 」 *

   ${tanda} ID transaksi : ${hasil.ref_id}
   ${tanda} Sn : Kode voucher atau token dikirim ke chat
  ${tanda} Produk : ${i.produk}
  ${tanda} Harga : ${i.harga}
```

Fungsi *ceklqbro* merupakan fungsi *broker listener* untuk platform LinkQu*,* yang digunakan untuk melakukan verifikasi status pembayaran secara berkala, ketika pembayaran diterima dan mendapatkan return *success* dari *broker* LinkQu, transaksi akan dilanjutkan pada platform Digiflazz menggunakan fungsi *cekTrxDF*.

Segmen Program 4.11 Fungsi transaksi Digiflazz

```
    const koneksiDF = new Digiflazz(
      process.env.DIGIFLAZZ_USERNAME,
      process.env.DIGIFLAZZ_PRODUCTION_KEY
    );
    exports.cekTrxDF = async (sku, tujuan, refId) => {
      try {
        const result = await koneksiDF.transaksiDF(sku, tujuan, refId);
        if (result.status === 'Pending') {
          await new Promise((resolve) => setTimeout(resolve, 10000));
          return await this.cekTrxDF(sku, tujuan, refId);
        } else
          return result;
      } catch (error) {
        return error;
      }
    };
```

Fungsi *transaksiDF* digunakan untuk melakukan *execute API* Digiflazz, untuk melakukan pembelian produk dengan memasukkan *parameters* kode sku produk, ID tujuan, dan nomor referensi. Fungsi *cekTrxDF* merupakan fungsi rekursif yang digunakan untuk memeriksa status transaksi pada digiflazz dimana pemeriksaan dilakukan setiap 10 detik.

4.3.7 Fungsi Update Produk

Segmen Program 4.12 Fungsi Update Produk

```
{
        async function simpanDataKeFile() {
          try {
            const dataks = await koneksiDF.daftarHargaDF();
            const filteredDataks = dataks.filter((item) => item.buyer_product_status);

            console.log(filteredDataks);
            const dataJSON = JSON.stringify(filteredDataks, null, 2);
            fs.writeFileSync('product.json', dataJSON);
            console.log('Data berhasil disimpan ke product.json');
            m.reply('Sukses Update Harga');
          } catch (error) {
            console.error('Gagal menyimpan data:', error);
            m.reply(`Gagal: ${error.message || error}`);
          }
        }

        simpanDataKeFile();
      }
```

Fungsi update produk merupakan fungsi yang hanya dapat di eksekusi oleh pemilik bot, fungsi ini akan melakukan *execute API* untuk mendapatkan list produk yang telah di konfigurasi pada platform Digiflazz, setiap perubahan akan di simpan pada *file product.json*.

4.3.8 Fungsi Update Rate

Segmen Program 4.13 Fungsi Update Rate

```
{
	function bacaData() {
	  return JSON.parse(fs.readFileSync('./database/rate.json'));
	}

	function tulisData(data) {
	  fs.writeFileSync('./database/rate.json', JSOxN.stringify(data, null, 2));
	}

	function ubahSemuaData(tipe, nilaiBaru) {
	  let rate = bacaData();

	  if (rate[tipe]) {
	    for (let properti in rate[tipe]) {
	      rate[tipe][properti] = parseFloat(nilaiBaru);
	    }
	    m.reply(`Data pada ${tipe} berhasil diubah: ${JSON.stringify(rate[tipe])}`);

	    tulisData(rate);
	  } else {
	    m.reply(`Error: Tidak dapat menemukan tipe ${tipe}`);
	  }
	}
	ubahSemuaData('member', parseFloat(args[0]));
}
```

4.3.9 Fungsi Isi Saldo Digiflazz

Segmen Program 4.14 Fungsi isi Saldo Digiflazz

```
{
	if (!isCreator) return m.reply(mess.pemilik);
	const jumlah = text.split(' ')[0];
	const bank = text.toUpperCase().split(' ')[1];
	let nama1 = text.toUpperCase().split(' ')[2];
	let nama2 = text.toUpperCase().split(' ')[3];
	let nama3 = text.toUpperCase().split(' ')[4];
	if (nama2 === undefined) {
	  nama2 = ' ';
	} else {
	  nama2;
	}
	if (nama3 === undefined) {
	  nama3 = ' ';
	} else {
	  nama3;
```

```
        }
        const nama = nama1 + ' ' + nama2 + ' ' + nama3;
        for (const i of rekBayar) {
          if (i.brand === 'BCA') {
            namaRek = i.atasNama;
            break;
          }
        }
        if (isNaN(parseInt(jumlah)))
          return (
            m.reply(
                `Minimal jumlah isi saldo digiflazz 200000 dan hanya angkanya saja tanpa titik, hanya tinggal copy dan paste pilihan yang telah disediakan dan sesuaikan nama pemilik rekeningnya`
            ),
            setTimeout(() => m.reply(`${command} 200000 BCA ${namaRek}`), 1000),
            setTimeout(() => m.reply(`${command} 200000 MANDIRI ${namaRek}`), 1100),
            setTimeout(() => m.reply(`${command} 200000 BRI ${namaRek}`), 1200)
          );
        if (jumlah < 200000) return m.reply('Minimal jumlah isi saldo digiflazz 200000');
        const deposit = await koneksiDF.depositDF(parseInt(jumlah), bank, nama);
        const desk = `* 「 ISI SALDO DIGIFLAZZ VIA ${bank} 」 *

${tanda} Jumlah transfer : ${rp(parseInt(deposit.amount))}
${tanda} Berita transfer : ${deposit.notes}
${tanda} Nomor rekening transfer : 6042 8888 90
${tanda} Nama Rekening : PT DIGIFLAZZ INTERKONEKSI INDONESIA

${botName}`;
        const deskss = [desk, rp(parseInt(deposit.amount)), deposit.notes];
        const messagess = deskss.map((desk) => ({
          text: desk,
          quoted: m,
        }));
        for (let i = 0; i < messagess.length; i++) {
          await vbot.sendMessage(from, messagess[i]);
        }
      }
```

Fungsi ini digunakan untuk melakukan *execute API* Digiflazz untuk melakukan pengisian saldo, fungsi ini hanya dapat diakses oleh pemilik bot menggunakan verifikasi dengan fungsi *isOwner.* Untuk melakukan pengisian saldo diperlukan memasukkan *parameters* jumlah saldo yang ingin di deposit, nama bank dan nama pemilik rekening, setelah itu API Digiflazz akan memberikan return

jumlah nominal transfer beserta kode unik dan berita acara transfer, hal ini digunakan platform Digiflazz untuk memverifikasi pembayaran secara otomatis.

4.3.10 Fungsi Cek Saldo Digiflazz

Segmen Program 4.15 Fungsi Cek Saldo Digiflazz

```
async cekSaldoDF() {
 const sign =
crypto.createHash('md5').update(`${this.username}${this.apikey}depo`).digest('hex');
      const data = {
         cmd: 'deposit',
         username: this.username,
         sign,
      };
      const options = {
         method: 'POST',
         body: JSON.stringify(data),
         headers: { 'Content-Type': 'application/json' },
      };
      try {
         const resp = await fetch(`${this.endpoint}cek-saldo`, options);
         const respData = await resp.json();
         return respData.data;
      } catch (err) {
         throw new Error(err);
      }
   }
```

Fungsi ini digunakan untuk menampilkan nominal saldo digiflazz yang dimiliki saat ini. Fungsi ini akan melakukan *execute API* dengan memberikan *body* sesuai dengan dokumentasi yang telah disediakan oleh platform. Hasil dari eksekusi API ini akan berbentuk JSON berisi nominal saldo yang dimiliki.

4.3.11 Fungsi Cek Saldo LinkQu

Segmen Program 4.16 Fungsi Cek Saldo LinkQu

```
async cekSaldoLQ() {
      const url = `${this.endpoint}linkqu-partner/akun/resume?username=${this.user}`;

      try {
         const resp = await fetch(url, { headers: this.headers });
         const respData = await resp.json();
         return respData.data;
      } catch (err) {
         throw new Error(err);
      }
   }
```

Fungsi ini digunakan untuk menampilkan nominal saldo LinkQu yang dimiliki saat ini. Fungsi ini akan melakukan *execute API* dengan memberikan *params* pada *URL* sesuai dengan dokumentasi yang telah disediakan oleh platform. Hasil dari eksekusi API ini akan berbentuk JSON berisi nominal saldo yang dimiliki.

4.3.12 Fungsi Verifikasi ID

Segmen Program 4.17 Fungsi Verifikasi ID

```
exports.cekID = async (id, game) => {
 if (game === 'Valorant') {
  const regex = /#(?<!\s)/;
  if (regex.test(id)) {
   try {
    const encodedID = encodeURIComponent(id);
    console.log(encodedID);
    const resp = await axios.get(`${cekIDValo}${encodedID}`);
    if (resp.data.success === true) {
     return resp.data;
    }
   } catch (err) {
    console.log(err.message);
    return {
     success: false,
    };
   }
  }

  return {
   success: false,
  };
 } else if (game === 'MOBILE LEGENDS') {
```

```
  const length = id.length;

  const id4 = id.slice(0, length - 4);
  const id5 = id.slice(0, length - 5);
  const zone4 = id.slice(-4);
  const zone5 = id.slice(-5);

  try {
   const resp = await axios.get(`${cekIDML}${id5}&zone=${zone5}`);
   if (resp.data.success === true || resp.status === 200) {
    return resp.data;
   }
  } catch (err) {
   console.log('Failed fetch ID for zone 5');
  }

  try {
   const resp = await axios.get(`${cekIDML}${id4}&zone=${zone4}`);
   if (resp.data.success === true || resp.status === 200) {
    return resp.data;
   }
  } catch (err) {
   console.log('Failed fetch ID for zone 4');
  }
  return {
   success: false,
  };
 } else {
  return {
   success: false,
  };
 }
```

Fungsi verifikasi ID di perlukan untuk melakukan verifikasi pada ID yang di input oleh pembeli, fungsi ini dijalankan sebelum proses transaksi dilakukan. Dalam melakukan verifikasi akan dilakukan *execute API* menggunakan library *axios* dengan parameters jenis game dan id, jika ID dapat ditemukan maka API akan memberikan return detail ID tersebut, namun jika tidak ditemukan akan memberikan return *success* dengan nilai *false.*

4.3.13 Fungsi Error Handler

Segmen Program 4.18 Error Handler

```
try {
 // Existing Code
} catch (err)
console.log(err)
await vbot.sendMessage(`${pemilik}@s.whatsapp.net`, {
    text: `*Laporan Error :*\n\n${util.format(err)}`,
  });
```

Untuk melakukan *error handling* digunakan *try catch statement,* dimana jika terjadi kesalahan atau error pada saat melakukan eksekusi kode yang terdapat pada *block try* akan dilakukan *throw Error* pada *block catch*, pada fungsi ini, *error handling* yang dilakukan adalah dengan menampilkan error pada *console*, dan mengirimkan pesan *error* pada pemilik / admin dari bot.

4.3.14 Fungsi Stress Test

Segmen Program 4.19 Stress Test

```
const { default: makeWASocket, initAuthCreds, useMultiFileAuthState } =
require('@whiskeysockets/baileys');
    const { performance } = require('perf_hooks');
    const fs = require('fs').promises;

    const handleMessage = () => {};

    const results = [];

    describe('WhatsApp Bot Stress Test', () => {
      let socket;
      let authState;

      beforeAll(async () => {
        const authDir = './session';
        await fs.mkdir(authDir, { recursive: true }).catch(() => {});

        try {
          const { state, saveCreds } = await useMultiFileAuthState(authDir);
```

```
      authState = { state, saveCreds };
      socket = makeWASocket({ auth: authState });
    } catch (error) {
      console.error('Failed to initialize authentication state:', error);
    }
  });

  const simulateIncomingMessages = async (numMessages) => {
    const startTime = performance.now();
    const messages = Array.from({ length: numMessages }, (_, i) => ({
      key: { remoteJid: 'test@wa', fromMe: false },
      message: { conversation: `Test message ${i + 1}` },
    }));

    await Promise.all(messages.map(handleMessage));

    const endTime = performance.now();
    return endTime - startTime;
  };

  const testResponseTime = async (numMessages) => {
    const responseTime = await simulateIncomingMessages(numMessages);
    results.push({ total_messages: numMessages, avg_time: `${responseTime.toFixed(2)}
ms` });
  };

  test('Average Response time', async () => {
    await testResponseTime(Math.pow(10,0)); //10^0
    await testResponseTime(Math.pow(10,1)); //10^1
    await testResponseTime(Math.pow(10,2)); //10^2
    await testResponseTime(Math.pow(10,3)); //10^3
    await testResponseTime(Math.pow(10,4)); //10^4
    await testResponseTime(Math.pow(10,5)); //10^5
    await testResponseTime(Math.pow(10,6)); //10^6
```

```
  });

  afterAll(() => {
    console.table(results, ['total_messages', 'avg_time']);
  });
});
```

Pengujian *stress testing* dilakukan untuk mengukur kemampuan bot Whatsapp dalam kondisi percobaan penggunaan oleh banyak *user* dalam satu waktu yang sama. Pengujian ini dilakukan dengan menggunakan *library* Jest, pengukuran dilakukan dengan mengambil nilai rata-rata / *average* waktu dari pesan awal dikirimkan oleh user hingga bot mengirimkan respon pesan dengan asumsi tidak ada jeda/*delay* yang di sebabkan oleh gangguan internet dari bot atau user.